Paper 1135-2017

The Ins and Outs of %IF

M. Michelle Buchecker, ThotWave Technologies, LLC.

ABSTRACT

Have you ever had your macro code not work and you couldn't figure out why? Even something as simple as %if &sysscp=WIN %then LIBNAME libref 'c:\temp'; ?

This paper is designed for programmers who know %LET and can write basic macro definitions already. Now let's take your macro skills a step farther by adding to your skillset. The %IF statement can be a deceptively tricky statement due to how we are used to thinking IF statements are processed in the DATA step and how that differs from how %IF statements are processed by the macro processor. Focus areas of this paper will be:

- Emphasizing the importance of the macro facility as a code generation facility
- How a DATA step IF statement differs from a macro %IF statement and when to use which
- Why semicolons can be misinterpreted on a %IF statement

INTRODUCTION

The macro facility is a text processing language. It is a language unto itself with its own rules and syntax. The results of the macro language are often used to generate SAS programming statements.

The first rule that cannot be emphasized enough is **hardcode everything first**. To make sure your macro program is successful, you first need to make sure that the SAS code you want to generate is correct. For example, LIBNAME libref 'c:\temp';

If the code doesn't work when you hardcode it, it's certainly not going to work when you add macro code.

The purpose of this paper is to figure out after you hardcode your program when to use a %IF statement or a regular IF statement. This will save you countless debugging hours.

WHEN TO USE IF VS. %IF

There are a lot of subtleties when it comes to differentiating when to use a SAS IF statement versus a macro %IF statement.

Table 1 IF vs. %IF

Here is a handy chart.

Use SAS IF	Use macro %IF
Condition is testing on a DATA step variable	Need to conditionally execute macro code
Need to conditionally execute SAS code	Need to conditionally send SAS code to the compiler
	Outside of a DATA step

Note what is missing from the table:

1. **Condition is testing on a macro variable**. Just because you are testing on a macro variable does not necessarily mean you should be using %IF in that situation. It depends on what that macro variable resolves to as well as what action you are performing if the condition is true.

2. **Inside of a DATA step**. Just because you are inside of a DATA step does not necessarily mean you should be using SAS IF in that situation. It depends on what the condition is, as well as what action you are performing if the condition is true.

CONDITIONAL TESTING ON A DATA STEP VARIABLE

Remember tip #1, **Hardcode everything first and make sure it works**. For example:

```
data test;
  set sashelp.heart;
  if cholesterol > 200 then status='Follow up';
run;
```

Now let's say we created two macro variables:

```
%let factor=cholesterol;
```

```
%let cholesterol=200;
```

Tip #2 is **punch holes in the program and substitute macro variable references**. Since we hardcoded a SAS IF statement to test a DATA step variable, we continue to use a DATA step IF statement. We just substitute what we hardcoded.

```
data test;
  set sashelp.heart;
  if &factor > &&&factor then status='Follow up';
Run;
```

We've met the rules from the above chart: testing on a DATA step variable, conditionally executing SAS code.

CONDITIONALLY EXECUTE MACRO CODE

Next let's discuss **Need to conditionally execute macro code** and **Outside of a DATA step**.

The **Outside of a DATA step** is the easy one. If you want to execute a PROC step based on a condition, than %IF is the way to go since you cannot embed a PROC step in a DATA step (CALL EXECUTE is an exception to this but is far more complicated).

For example:

```
%IF &sysday=Friday %THEN %DO;
   proc means data=sashelp.heart;
   run;
%END;
```

The **Need to conditionally execute macro code** gets slightly more complicated. Take the following incorrect example:

```
data test;
  if &sysday=Friday then do;
     %let status=Party!;
  end;
  else do;
     %let status=Not Friday;
  end;
run;
```

Regardless of the day of the week, the value of status will always be **`Not Friday`**. The reason is that macro code is processed during the tokenization phase. That's the phase where SAS scans the code, looks for syntax errors, and compiles the step. The DATA step won't execute until the step boundary (the

RUN statement in this case) is encountered.

Below, I've put a **C** next to a statement that is compiled only, and a **CE** next to a statement that is compiled and then immediately executed.

```
C data test;
C  if &sysday=Friday then do;
CE     %let status=Party!;
C  end;
C  else do;
CE     %let status=Not Friday;
C  end;
C run;
```

Which means the statements that got compiled first and immediately executed are:

```
%let status=Party!;
```

```
%let status=Not Friday;
```

If you just saw that code, you realize that the text **`Not Friday`** will always be the last value assigned to the macro variable status.

So what happens with the rest of the code? Well once the RUN statement is encountered, the statements that were previously just compiled are now executed. Here is that code:

```
data test;
  if &sysday=Friday then do;
  end;
  else do;
  end;
run;
```

There are no statements to execute regardless if the condition is true or false.

So essentially the **timing** is key here. Macro statements compile and execute as soon as they are encountered.

To fix this example

```
%if &sysday=Friday %then
     %let status=Party!;
%else
     %let status=Not Friday;
```

WAIT!

What if you have to test on a DATA step variable, but then create a macro variable? Then use CALL SYMPUT.

CONDITIONALLY EXECUTE OR SEND SAS CODE

Remember the chart from above:

Use SAS IF	**Use macro %IF**
Condition is testing on a DATA step variable	Need to conditionally execute macro code
Need to conditionally execute SAS code	Need to conditionally send SAS code to the compiler

Use SAS IF	**Use macro %IF**
	Outside of a DATA step

Let's compare and contrast **Need to conditional execute code** vs. **Need to conditionally send code to the compiler**, as that is a challenge to understand.

The question I like to ask is, "will it hurt if the compiler sees this code?". If the answer is no, then use a SAS IF statement. If the answer is yes, then use a %IF statement.

A good example of this is conflicting statements in the DATA step, such as KEEP statements. If the compiler sees more than one KEEP statement in a step it uses the one with the most variables, which isn't what you want all the time (or you wouldn't have written the other KEEP statement).

Example of using both %IF and regular IF in the same step:

```
%macro mtest(type=yes);
data test;
  %if type=yes %then %do;
      keep chol_status sex weight height total;
  %end;
  %else %do;
      keep sex weight height;
  %end;
  set patient_info;
  if chol_status ne ' ' then
    Total=LDL+HDL;
Run;
%mend;
```

SYNTAX OF %IF STATEMENT WHEN SENDING SAS CODE TO COMPILER

Because the macro facility is a language that generates code in another language, you have to make sure that the macro language has the correct syntax and the generated code has the correct syntax.

Syntax for macro %IF:

```
%IF condition %THEN action;
```

Or

```
%IF condition %THEN %DO;

      action1;

      action2;

%END;
```

Notice that in the first %IF statement that there is a semicolon that the macro facility needs as part of the syntax for the %IF statement.

Going back to the original hardcoded example, LIBNAME libref 'c:\temp';   notice there is a semicolon that ends the SAS LIBNAME statement.

When combining these 2 pieces of code together, some people mistakenly code:

%if condition %then LIBNAME libref 'c:\temp';

There is only one semicolon. So who gets the semicolon? The macro language or the SAS language? What happened to the second semicolon? Remember, we are combining two languages here. The answer is the macro language steals that semicolon. Which means the SAS compiler never sees a semicolon to end the LIBNAME statement.

There are three solutions to this.

1. `%if condition %then LIBNAME libref 'c:\temp';;`

In this case the first semicolon is taken by the macro language for the %IF statement, and the second semicolon is passed to the SAS compiler as text. But doesn't it look weird? And if someone inherits this code, they may very likely delete that second semicolon thinking it is a typo.

2. `%if condition %then LIBNAME libref 'c:\temp'%str(;);` or `%if condition %then %str(LIBNAME libref 'c:\temp';);`

In this situation we still have two semicolons. By using the %STR macro quoting function, it tells the macro facility "Oh great macro processor: don't treat this semicolon inside the parenthesis as an end of statement, but instead treat it as constant text for you to ignore". So in this case this first semicolon will get passed to the SAS compiler, and the second semicolon is used by the macro facility. It does require use of a macro quoting function though. Quoting functions are complex and may not be suited for inexperienced macro programmers.

3.
```
%if condition %then %do;
   LIBNAME 'libref c:\temp';
%end;
```

In this last solution we have the macro facility perfectly happy with the semicolons for the %do and %end statements, and the code inside the %DO block will get passed to the SAS compiler. It is a normal-looking statement which makes for easy readability. Now traditional SAS programmers are not necessarily used to having just one action statement inside of a DO block, so this may look peculiar to them.

As you can tell, there are pros and cons with each solution. There isn't one "best" answer, as it is a personal preference. Although I think it is safe to say that solution one is generally the least preferable. I'm a fan of solution two, and yet my manager tends to code solution three. So it really is up to you.

OTHER DIFFERENCES

Because %IF is part of the macro facility, it does not have the robustness that the DATA step IF has when it comes to calculations. For example, the DATA step can easily process IF 10 < var < 25, meaning var is between 10 and 25.

However, %IF looks at each component separately. So %IF 10 < &var <25 will always evaluate to TRUE (1) regardless of the value of &var. Let's say &var has a value of 50. The macro facility will first evaluate 10 < 50. The result is TRUE or in the SAS world, 1. It then takes that value, 1, and compares it to 25, e.g. 1 < 25, and that answer is TRUE. The fix is to separate the expression to %IF 10 < &var and &var < 25.

Ultimately remember that %IF is a more simplistic form of IF when it comes to doing math.

CONCLUSION

%IF is part of the macro facility. The purpose of the macro facility is a text processing facility to generate and manipulate SAS code. Don't forget to hardcode everything first. Next. strategically think about what is executable SAS code, and use IF statements on that. Lastly, determine what code may need to be conditionally submitted to the SAS compiler, or conditionally executed by the macro processor and that is more appropriate for %IF.

ACKNOWLEDGMENTS

I wouldn't have the macro skills I do if it weren't for the expertise of Warren Repole.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Michelle Buchecker
ThotWave Technologies
mbuchecker@thotwave.com